공학석사 학위논문

효율적인 맵리듀스 기반 문자열 유사도 조인

Efficient String Similarity Joins using MapReduce

2015년 7월

서울대학교 대학원

전기·정보공학부

이 창 형

효율적인 맵리듀스 기반 문자열 유사도 조인

지도교수 심 규 석

이 논문을 공학석사 학위논문으로 제출함
2015년 7월

서울대학교 대학원
전기·정보공학부
이창형

이창형의 공학석사 학위논문을 인준함
2015년 7월

위 원 장 김태환 (인)

부위원장 심규석 (인)

위 원 홍성수 (인)

초 록

문자열 유사도 조인은 데이터 베이스 분야에서 매우 중요하고 자주 사용되는 질의이다. 최근 토큰 기반 유사도와 문자 기반 유사도의 장점을 혼합한 Fuzzy 토큰 자카드 유사도가 제안되었다. 그러나 Fuzzy 토큰 자카드 유사도를 이용한 조인은 수행 시간이 너무 오래 걸려 이를 대용량 데이터에서도 사용하기는 어려웠다. 따라서 이를 극복하기 위해 맵리듀스 프레임워크를 이용하는 새로운 분산병렬처리 알고리즘과 이를 위한 새로운 시그니쳐를 제안하였다. 그리고 기존의 단일 머신 알고리즘과 실험을 통해 그 성능을 비교하였으며 20대의 컴퓨터를 이용하였을 때 최대 7배까지 성능이 향상되는 것을 확인할 수 있었다. 또한 컴퓨터의 수를 늘렸을 때 분산처리 방식의 유사도 조인 알고리즘 수행시간이 효과적으로 줄어드는 것을 확인하였다.

주요어 : 문자열, 유사도 조인, 맵리듀스, 알고리즘, 하둡
학 번 : 2013-20864

목 차

제 1 장 서 론

제 1 절 연구의 배경 및 내용

문자열 유사도 조인은 두 문자열 집합 사이에서 유사한 문자열 쌍을 찾아내는 작업으로, 데이터 통합이나 협업 필터링 등에서 필수로 요구되기 때문에 데이터베이스 분야에서 매우 자주 사용되며 중요한 일이다. 예를 들어 인터넷 검색 엔진의 검색 쿼리 데이터를 이용해 문자열 유사도 조인을 수행한다면 어떤 사용자들이 비슷한 쿼리를 검색했는지 알 수 있다. 이를 활용하면 수많은 사용자를 유사한 사용자들의 집단으로 나누어 분류할 수 있게 되며, 집단의 특성을 파악해 검색 결과 표시나 광고 추천을 더 효율적으로 할 수 있다. 또한 웹 상에 흩어져있는 데이터들을 통합하려고 할 때, 다양하고 복잡한 형태로 흩어져 있는 수많은 데이터들 사이에는 오탈자 뿐만 아니라 단어의 순서 또한 섞여있는 경우가 많다. 따라서 그림1과 같이 비슷한 경우에도 통합이 가능해야 하기에 문자열 유사도 조인이 반드시 필요하다. 이 밖에도 다양한 방면의 활용이 가능하기에 기존에 많은 연구들이 진행되었으며 여러 알고리즘들이 제안되었다.

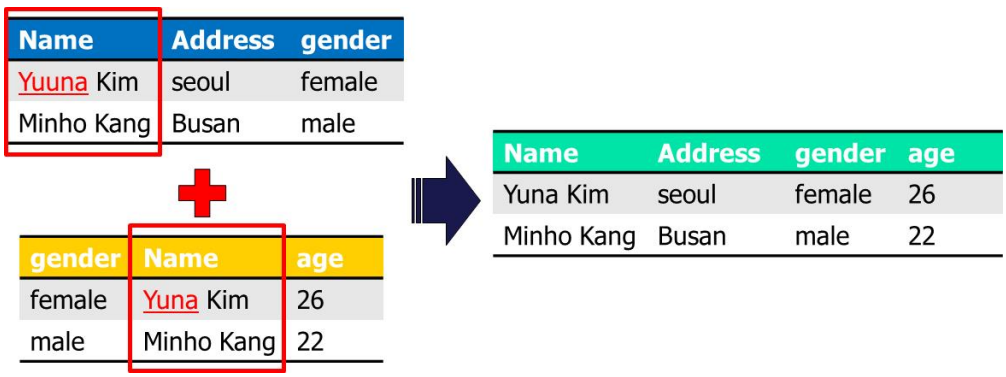

그림 1 데이터 통합에서의 문자열 유사도 조인

문자열 유사도 조인을 수행하기 위한 문자열 유사도는 크게 토큰(token) 기반 유사도와 문자(character) 기반 유사도로 나누어진다. 그 중에서 토큰 기반 유사도 함수는 문자열을 단어 단위로 잘라 토큰 집합으로 만든 후 자카드(Jaccard) 유사도나 코사인(cosine) 유사도 등을 사용해서 계산하는데, 이 방식은 오탈자가 있는 단어나 같은 의미를 갖는 비슷한 단어들은 다른 단어로 취급하는 문제가 있다. 이에 반해 문자 기반 유사도는 문자열 두 개 사이에 다른 문자가 얼마나 있는지를 측정하는 방식을 사용한다. 대표적으로 편집 유사도(edit similarity)를 이용하는 방식이 있는데, 이러한 방법으로는 “tv show”, “show tv”와 같이 단어의 순서가 바뀐 경우에 비슷한 문자열임에도 불구하고 찾아내지 못한다는 단점이 있다. 따라서 이 단점을 해결하는 혼합 유사도[1]가 제안되었는데 이 혼합 유사도는 유사도 계산의 비용이 매우 크다는 단점을 가지고 있다. 따라서 대용량의 데이터에는 기존의 단일 머신 알고리즘[1]을 사용할 수 없기에 이를 분산처리 시스템 중 하나인 맵리듀스를 이용한 새로운 알고리즘을 제안하고, 실험을 통해 이 알고리즘의 효율성을 보이고자 한다. 본 논문의 기여는 다음과 같다.

- 기존의 시그니쳐 방식보다 후보 레코드 쌍을 적게 만드는 새로운 방식의 시그니쳐를 제안하여 유사도 조인의 연산 횟수를 줄였다.
- 맵리듀스를 이용한 분산 병렬 처리 알고리즘을 제안하여 대용량의 데이터도 처리할 수 있게 하였으며, 유사도 조인을 위해 사용하는 컴퓨터의 수를 늘리면 선형적으로 성능이 향상되기 위한 작업 분배 방법을 제안하고 실험을 통해 그 확장성(scalability)을 보였다.

이후 본 논문은 다음과 같은 구성으로 이루어져 있다. 2장에서는 분산 병렬 처리와 문자열 유사도 조인에 대한 관련 연구를 소개하고 정리하며 3장에서는 본 논문의 분산 처리 알고리즘과 내용을 설명한다. 4장에서는 실험을 통해 본 논문의 알고리즘과 기존의 알고리즘의 수행시간을 측정하고 비교한다. 그리고 5장에서는 내용을 정리하여 결론을 내린다.

제 2 장 관련 연구

제 1 절 문자열 유사도

토큰 기반 유사도는 문자열을 단어 단위로 잘라 토큰의 집합으로 만들고 이들 간에 공통되는 토큰이 많을수록 유사도 값이 높아지는 함수이다[1]. 대표적으로 자카드 유사도와 코사인 유사도가 있다. 예를 들어 r은 "todai weather korea"를 나타내는 문자열이고 s는 "corea weather today"라는 문자열이라고 하자. 이를 이용해 자카드 유사도를 계산해보면 식(2-1)과 같다. 여기서 tr과 ts는 각각 r과 s을 이용해 만든 토큰 집합을 말한다.

$$Jac(r,s) = \frac{|tr \cap ts|}{|tr| + |ts| - |tr \cap ts|} = \frac{1}{3+3-1} = 0.2 \qquad \text{식(2-1)}$$

이 방법의 경우, r과 s는 비슷한 문자열임에도 불구하고 "todai"와 "today", "corea"와 "korea"는 서로 다른 토큰이기 때문에 유사도가 높게 나타나지 않는 단점이 있다.

문자 기반 유사도 중에서는 편집 거리(edit distance)와 편집 유사도가 가장 많이 사용된다[1]. 편집 유사도는 비교할 두 문자열 r,s에 대해 r을 s로 바꾸는데 필요한 최소의 문자 수정(삽입, 삭제, 대체)개수인 편집 거리를 이용하여 정의된다. 그림2의 예제를 보면 "todai"를 "corea"로 바꾸는 데는 4번의 대체가 필요하고 "korea"를 "today"로 바꿀 때도 마찬가지로 4번의 대체가 필요하다 따라서 r을 s로 바꾸기 위해서는 총 8번의 대체가 필요하여 편집 거리는 8이 계산된다. 따라서 r과 s사이의 편집 거리와 편집 유사도는 다음과 같다.

$$Edit\ Distance(r,s) = ED(r,s) = 8 \qquad \text{식(2-2)}$$

$$Edit\ Similarity(r,s) = ES(r,s) = 1 - \frac{ED(r,s)}{max(|r|,|s|)} = \frac{11}{19} \approx 0.58 \qquad \text{식(2-3)}$$

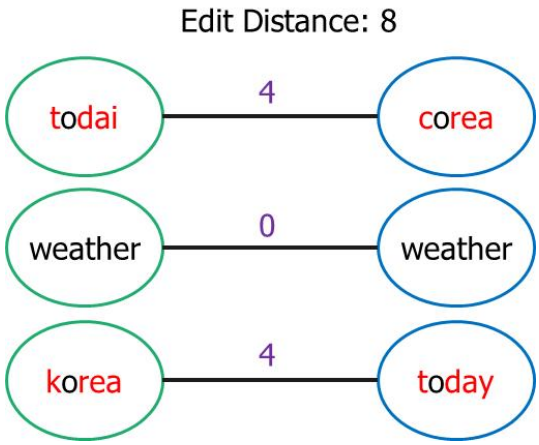

그림 2 r과 s사이의 편집 거리

식(2-3)과 같이 문자 기반 유사도는 순서가 바뀐 문자열을 전혀 고려하지 못하고 낮은 유사도가 계산되는 것을 볼 수 있다.

이에 토큰 기반 유사도와 문자 기반 유사도의 단점을 극복하기 위해 두 방법을 결합시킨 유사도가 혼합 유사도이다. 최근에 Fuzzy 토큰 자카드 유사도가 새롭게 제안되었는데 이 유사도는 다음과 같은 방법으로 계산한다.

① 비교하려는 문자열 r과 s를 단어 단위로 잘라 토큰 집합 tr과 ts로 만든다.

② 두 개의 토큰 집합 사이 모든 토큰 쌍의 편집 유사도를 계산하여 주어진 한계값 δ를 넘는 쌍만 남기고 이들의 집합을 E라고 한다.

③ ①에서 만든 tr과 ts에 속해있는 토큰들을 정점으로 하며 E에 존재하는 토큰 쌍들을 간선으로 연결하고, 그 간선들의 가중치는 토큰 쌍의 편집유사도로 하는 이분 그래프(bipartite graph) $G = (tr, ts, E)$를 만든다.

④ 이분 그래프 G에서 그림3에서와 같이 maximal matching을 찾아 무게 값을 합한 후 이 값을 $Fuzzy\ overlap(FOV)$ 이라 한다. 여기서

maximal matching이란 정점이 겹치지 않게 간선들을 선택하였을 때 그 가중치 값의 합이 최대가 되도록 하는 간선들의 집합이다.

⑤ Fuzzy 토큰 자카드 유사도 함수는 자카드 유사도 함수의 $|tr \cap ts|$를 FOV로 교체한 것으로 그 정의는 다음과 같다.

$$\mathrm{F}uzzyJac(r,s) = \frac{FOV}{|tr| + |ts| - FOV} \qquad \text{식(2-4)}$$

예를 들어 편집 유사도 한계 값을 0.2로 할 때 r과 s사이의 Fuzzy 토큰 자카드 유사도를 계산해보자. 먼저 r과 s를 이용해 이분 그래프를 만들면 그림3의 왼쪽 그래프 같이 그려진다. 여기서 maximal matching을 찾게 되면[8] 각 정점에서 가중치가 최대인 간선들을 선택하게 되고 그림3의 오른쪽 그래프와 같이 선택하게 된다. 이를 이용해 Fuzzy 토큰 자카드 유사도를 계산해보면 다음과 같다.

$$FuzzyJac(r,s) = \frac{FOV}{|tr| + |ts| - (FOV)} = \frac{\frac{4}{5} + 1 + \frac{4}{5}}{3 + 3 - (\frac{4}{5} + 1 + \frac{4}{5})} \qquad \text{식(2-5)}$$

$$\approx 0.76$$

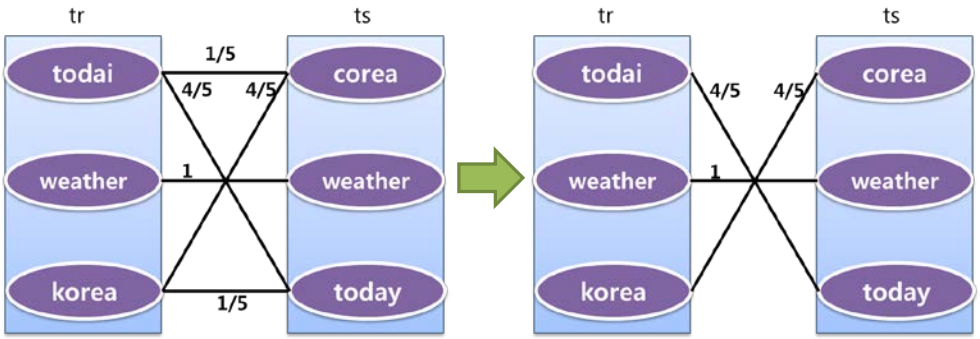

그림 3 r과 s사이의 maximal matching

제 2 절 문자열 유사도 조인

분할 기반 문자열 유사도 조인[3]은 편집 거리를 유사도 함수로

하는 조인을 효율적으로 처리하기 위한 알고리즘이다. 비교할 문자열 두 개 사이의 편집거리가 λ이하라면 문자열 하나를 $\lambda+1$개의 조각으로 나누었을 때, 그 중 한 조각은 비둘기 집의 원리에 의해 반드시 나머지 문자열 하나의 일부분과 일치한다. 그렇기에 모든 문자열을 $\lambda+1$개의 조각으로 나눈 후, 그 조각들을 각 문자열에 일치하는 부분이 있는지 확인해본다. 이 때 일치하는 부분이 있을 경우, 두 문자열 사이의 유사도를 직접 계산하여 검증하는 방법이다. 일치하는 부분이 없는 문자열 쌍에 대해서는 유사도 연산을 하지 않아도 되므로 연산량을 줄일 수 있다.

맵리듀스 기반 문자열 유사도 조인[4]은 분할 기반 문자열 유사도 조인[3]을 맵리듀스를 이용한 방법으로 바꾼 알고리즘이다. [3]에서와 같이 문자열을 여러 조각으로 나눈 후 이를 여러 대의 컴퓨터로 나누어 병렬적으로 유사도를 계산할 수 있게 만들었다.

Fuzzy 토큰 기반 유사도 조인[1]에서는 Fuzzy 토큰 유사도를 소개하고 Fuzzy 토큰 유사도를 이용한 유사도 조인을 제안한다. 이 조인 알고리즘에서는 각 문자열에서 문자열의 일부를 시그니쳐(signature)로 만들어 저장한 후, 이를 다른 문자열의 시그니쳐와 비교해 본다. 만약 두 시그니쳐 사이의 편집 거리가 1이하일 경우, 두 문자열이 비슷할 가능성이 있다고 판단하고 직접 유사도를 계산한 후 한계값 조건을 만족한다면 결과로 내보내는 알고리즘이다. 그러나 이 알고리즘의 경우 모든 시그니쳐를 메모리에 올려놓고 알고리즘을 수행해야 하기 때문에 분산병렬처리가 불가능하다. 따라서 대용량의 데이터에는 적용이 불가능한 알고리즘이라는 단점이 있다.

제 3 절 분산 병렬 처리

처리해야 하는 데이터의 양이 급증함에 따라 한 대의 컴퓨터로 모든 데이터를 처리하기가 어려워졌다. 이에 구글은 여러 대의 컴퓨터를 이용해 데이터를 분산 병렬처리하는 프레임워크인 맵리듀스(MapReduce)[2]를 개발하였다. 맵리듀스는 Map함수와 Reduce함수로 구성되어 있으며 맵리듀스의 작업이 시작되면 각 데이터 조각들은 키-값 쌍 <k1,v1>의 형태로 변환되어 Map 함수의 입력으로 들어간다. 이후 사용자가 구현한 Map 함수의 작업을 처리하고 그 결과를 다시 키-값 쌍 <k2,v2>의 형태로 만들어 내보내게 된다. 이후 프레임워크가 같은 키를 가진 키-값 쌍들이 한 대의 컴퓨터로 모이도록 나누어주고 Reduce 작업을 수행한다. Reduce 함수는 각 키마다 한번씩 호출되는데, 이때 해당 키와 쌍을 이루는 값들의 리스트 list(v2)를 함께 입력으로 받아 호출된다. 이후 사용자가 구현한 Reduce 함수의 내용대로 값들을 처리하여 키-값 쌍 <k3,v3>의 형태로 내보낸다. 단어의 개수를 세는 간단한 예제를 통해 맵리듀스의 과정을 그림4를 통해 설명해보면, 먼저 그림4의 왼쪽과 같이 입력 데이터가 주어지면 Map 함수에서는 각 단어를 키로 하고 단어의 개수를 값으로 해서 키-값 쌍 <단어, 개수>의 형태로 내보낸다. 같은 키를 가진 값들은 모여서 Reduce함수의 입력으로 들어간다. 그리고 Reduce함수에서는 이 값들을 모두 더해 키-값 쌍 <단어, 총 개수>의 형태로 결과는 내보낸다.

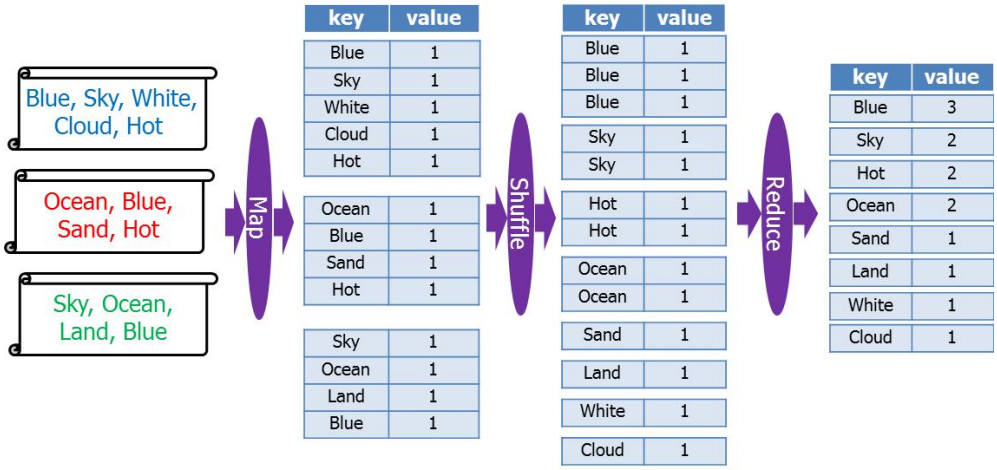

그림 4 맵리듀스를 이용한 워드 카운트 예제

맵리듀스 프레임워크는 스카인라인 질의[17], 클러스터링[18], 추천시스템[19] 등 다양한 분야의 연구[20]에서 활용되고 있으며 이중에서도 특히 맵리듀스를 이용해 조인 질의를 처리하는 연구는 숫자를 비교하는 동등조인[11]과 세타조인[9,10]에 대한 연구들과 문자열을 비교하는 문자열 유사도 조인[4,12,13,14,15,16]에 대한 연구들이 존재한다. 이 중에서 자카드 유사도를 이용한 유사도 조인에 대한 연구는 [4,12,14]에서 이루어졌으며 코사인 유사도를 이용한 연구는 [15,16]에서 이루어졌다. 맵리듀스의 오픈 소스 버전인 하둡(Hadoop)[7]이 널리 사용되며 본 논문의 실험에서도 이를 사용하였다.

제 3 장 분산 처리 유사도 조인

문자열 유사 조인 문제는 주어진 두 개의 문자열 집합 R과 S에서 두 문자열의 유사도 함수의 값이 한계점보다 크거나 같은 쌍을 모두 찾아내는 문제이다. 이때 유사도 함수 SIM과 한계값 θ가 주어지며 문자열 r과 s에 대해 $SIM(r,s) \geq \theta$를 만족하는 (r,s)쌍을 찾아내는 것이 목표이다. 이 때 유사도 함수는 Fuzzy 토큰 자카드 유사도를 사용한다. 따라서 조인을 하기 위해서는 편집 유사도의 한계값 δ와 유사도의 한계값 θ 두 가지가 필요하다.

본 논문에서 제안하는 유사도 조인 알고리즘은 유사도 연산의 횟수를 줄이기 위한 방법으로 레코드를 토큰 집합으로 만든 후, 이를 이용해 비교하는 두 문자열이 유사하다면 반드시 하나는 일치할 수밖에 없도록 이 논문에서 제안하는 시그니쳐 생성 방법을 통해 시그니쳐를 만들어서 내보낸다. 그리고 시그니쳐 사이에 일치하는 쌍을 찾는다면 그 시그니쳐를 포함하고 있는 문자열들을 여러 대의 컴퓨터에 분산시켜 동시에 유사도 연산을 수행하는 알고리즘이다. 간략한 방법은 다음과 같으며 자세한 내용은 다음 1절부터 5절에 걸쳐 설명한다.

1단계: 문자열 집합 R, S에 있는 단어의 수를 센다.

2단계: R과 S에 속한 문자열들의 토큰 중 일부를 시그니쳐로 만들어 내보낸다.

3단계: 기존의 연구를 이용해 유사한 시그니쳐의 쌍을 모두 찾는다.

4단계: 3단계에서 찾은 유사한 시그니쳐의 쌍을 이용해 유사할 것으로 예상되는 레코드 쌍을 찾고 그 수가 많다면 레코드 쌍의 집합을 여러 개의 조각으로 나누어 각 컴퓨터에 분배한다.

5단계: 4단계에서 각 컴퓨터에 모인 레코드 쌍들의 Fuzzy 토큰 자카드 유사도를 계산해 유사한지 확인한다.

제 1 절 토큰 빈도 카운팅

첫 번째 단계에서는 모든 레코드를 단어 단위로 잘라서 토큰으로 만든 후 각 토큰의 빈도 수를 센다. 이는 다음 단계에서 시그니쳐를 생성할 때 빈도 수가 적게 나타나는 토큰들로 시그니쳐를 생성해 연산량을 줄이기 위함이다.

토큰 빈도 카운팅 알고리즘의 의사코드를 그림6에 나타내었다. Map단계에서는 각 토큰을 키로 하고 값을 1로 하여 내보내고 Reduce단계에서 같은 토큰을 키로 하는 키-값 쌍을 모두 모아 값을 전부 더해서 내보내 각 토큰의 빈도 수를 계산한다. 그림5를 살펴보면 테이블 R의 첫 번째 레코드가 “top tv show”라는 문자열일 때, Map에서는 “top”, “tv”, “show”라는 토큰들로 나누어 각각의 개수를 값으로 하여 <top,1>, <tv,1>, <show,1>의 형태로 내보내게 된다. Reduce에서는 이를 모아 같은 토큰을 키로 하는 값들을 더하여 Map에서와 같이 <단어, 개수>의 형태로 내보내게 된다.

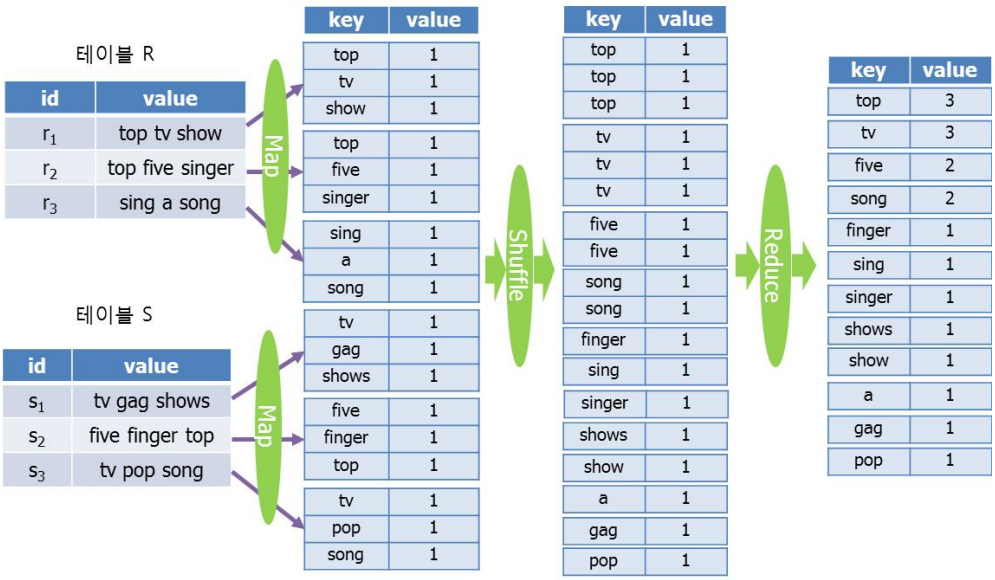

그림 5 토큰 빈도 카운팅의 예제

```
Fuction Token-Counting.map(<(db, index), (record)>)
begin
 1. Split record into token set T
 2. T = {t1,t2,...,t|n|}
 3. for ti ∈ T such that 1 ≤ i ≤ |n| do
 4.       emit<ti, 1>
end

Function Token-Counting.Reduce(<t, list(count)>)
begin
 1. sum=0
 2. for all count ∈ list(count) do
 3.      sum=sum+count
 4. emit<t, sum>
end
```

그림 6 토큰 빈도 카운팅 알고리즘의 의사코드

제 2 절 시그니쳐 생성

비교하려는 레코드 쌍 r과 s가 유사하다면, r과 s를 단어 단위로 잘라 만든 토큰 집합 tr과 ts 사이에는 반드시 유사한 토큰 쌍이 존재한다. 따라서 유사한 토큰 쌍을 가지고 있는 레코드 쌍을 찾아서 유사도를 계산해본다면 조인의 결과로 나오는 모든 레코드 쌍을 찾아낼 수 있다. 따라서 유사도 조인을 하기에 앞서 문자열 집합 R과 S의 모든

토큰들을 모아 유사한 토큰 쌍을 먼저 찾아내려 한다. 이 때 두 레코드가 유사하다면 유사도 조인의 정의에 따라 식(3-1)과 같은 부등식을 알 수 있고, 이를 FOV에 대하여 정리한다면 식(3-2)와 같다.

$$FuzzyJac(\mathrm{r},\mathrm{s}) = \frac{FOV}{|tr| + |ts| - FOV} \geq \theta \qquad \text{식 (3-1)}$$

$$FOV \geq \frac{\theta}{1+\theta}(|tr| + |ts|) \qquad \text{식 (3-2)}$$

만약 r이 s보다 토큰의 수가 적거나 같다면($|tr| \leq |ts|$이면) FOV의 최대값은 $|tr|$이 되고 위에 식에 대입한다면 아래와 같은 식이 나온다.

$$|tr| \geq FOV \geq \frac{\theta}{1+\theta}(|tr| + |ts|) \qquad \text{식 (3-3)}$$

$$|tr| \geq \theta \cdot |ts| \geq \theta \cdot |tr| \qquad \text{식 (3-4)}$$

식(3-4)를 다시 식(3-2)에 대입한다면 다음과 같다.

$$FOV \geq \frac{\theta}{1+\theta}(|tr| + |ts|) \geq \frac{\theta}{1+\theta}(\theta \cdot |ts| + |ts|) \geq \theta \cdot |ts| \geq \theta \cdot |tr| \qquad \text{식 (3-5)}$$

반대로 s의 토큰 수가 r의 토큰 수보다 적다면($|tr| > |ts|$이면) FOV의 최대값은 $|ts|$가 되고 다음과 같은 식을 얻을 수 있다.

$$|ts| \geq FOV \geq \frac{\theta}{1+\theta}(|tr| + |ts|) \qquad \text{식 (3-6)}$$

$$|tr| \geq |ts| \geq \theta \cdot |tr| \qquad \text{식 (3-7)}$$

식(3-7)을 다시 식(3-2)에 대입한다면 다음과 같다.

$$FOV \geq \frac{\theta}{1+\theta}(|tr| + |ts|) \geq \frac{\theta}{1+\theta}(|tr| + \theta \cdot |tr|) \geq \theta \cdot |tr| \qquad \text{식 (3-8)}$$

따라서 두 문자열이 유사할 경우, 식(3-5)와 식(3-8)을 통해 $FOV \geq \theta \cdot |tr|$를 만족해야 한다는 사실을 알 수 있다. 그러나 하나의 토큰 쌍의 편집 유사도 값은 1을 넘을 수가 없기 때문에 비슷한 레코드 r과 s사이에는 적어도 $\lceil \theta \cdot |tr| \rceil$개의 유사한 토큰 쌍이 존재해야 한다. 이를 이용해 r에서는 $(|tr| - \lceil \theta \cdot |tr| \rceil + 1)$개의 토큰을 시그니쳐로 내보내고

s에서는 토큰 전부를 시그니쳐로 내보낸다면, r의 시그니쳐 중 적어도 하나는 s의 시그니쳐와 유사하다. 만약 유사한 시그니쳐를 내보낸 레코드 쌍이 발견될 경우, 이를 후보 레코드 쌍이라 하고 직접 유사도 계산

Function Gen-sig.map(<(db, index), (record)>)

begin

1. Load **frequency list** from First Reduce
2. Split **record** into token set T
3. $T = \{t_1, t_2, ..., t_{|n|}\}$
4. Sort T in ascending order based on the **frequency list**
5. **If** db=="R" **then**
6. **for** $t_i \in T$ s.t. $1 \le i \le |n| - \lceil |\theta \cdot |n| \rceil + 1$ **do**
7. emit<t_i, (db, index)>
8. **If** db=="S" **then**
9. **for** $t_j \in T$ s.t. $1 \le j \le |n|$ **do**
10. emit<t_j, (db, index)>

end

Function Gen-sig.reduce(<t, list(db, index)>)

begin

1. P←new List
2. **for** all (db, index) ∈ list(db, index) **do**
3. append(P,(db,index))
4. emit<t, P>

end

그림 7 시그니쳐 생성 알고리즘의 의사코드

을 하게 된다. 그러나 입력으로 받은 문자열 집합에 자주 등장하는 토큰을 시그니쳐로 할 경우 후보 레코드 쌍의 수가 매우 커져서 유사도 계산의 양의 많아진다. 그렇기 때문에 유사한 시그니쳐 쌍의 수를 최대한 적게 하기 위해 1단계에서 센 토큰의 빈도 수를 이용하여 각 레코드의 토큰을 빈도의 오름차순으로 정렬하고 앞에서부터 시그니쳐로 내보낸다.

이를 이용한 시그니쳐 생성 알고리즘의 의사코드는 그림7과 같다. 먼저 Map단계에서는 각 레코드를 단어 단위로 잘라 토큰의 집합으로 바꾼 후, r 에서는 빈도가 적은 순서대로 $(|tr| - \lceil \theta \cdot |tr| \rceil + 1)$ 개의 토큰만을 시그니쳐로 만들어 내보내고 s에서는 모든 토큰을 시그니쳐로 만들어 내보낸다. 이렇게 내보낸 시그니쳐를 모았을 때 만약 r과 s가 유사하다면 둘 사이에 적어도 하나의 유사한 시그니쳐 쌍이 존재한다.

Reduce단계에서는 공통된 토큰을 시그니쳐로 내보낸 r과 s들을 모아 리스트를 만든다. 이를 통해 3단계에서 같은 토큰에 대한 편집 유사도 연산을 중복해서 하지 않도록 하여 전체 연산량을 줄일 수 있다. 그림8에서는 테이블 R과 S에서 각각 시그니쳐를 생성해 내보내는 예제를 보여준다. 편집 유사도 한계값 δ는 0.6으로 하고 유사도 한계값 θ가 0.8이라 했을 때, Map단계에서 테이블 R의 레코드는 모두 3개의 토큰을 포함하고 있기 때문에 $(|tr| - \lceil \theta \cdot |tr| \rceil + 1)$은 1이 되어 한 개의 토큰만을 시그니쳐로 내보내게 된다. 이 때, 테이블 R의 첫 번째 레코드인 “top tv show”에서 빈도수가 가장 적은 “show”을 시그니쳐로 만들어 키로 설정하고 값에는 레코드의 id를 넣어 내보낸다. 그리고 앞에서 설명한 바와 같이 테이블 S의 레코드는 모든 토큰을 시그니쳐로 내보내는 것을 확인할 수 있다. 그리고 Reduce단계에서 이를 모아 각 시그니쳐를 내보낸 레코드의 리스트를 만든다.

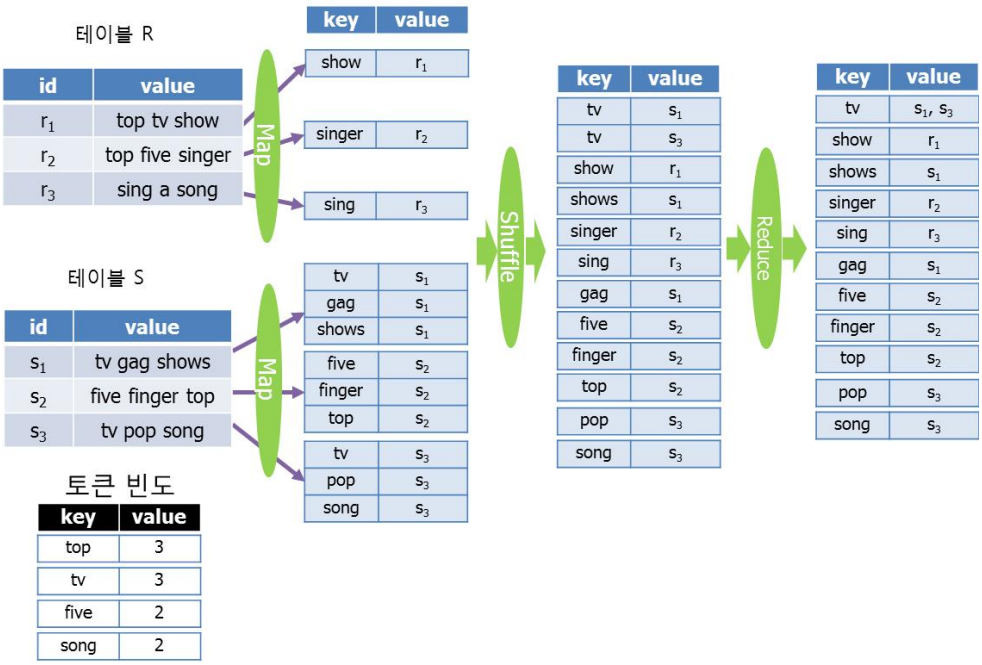

그림 8 시그니쳐 생성 알고리즘의 예제

제 3 절 문자 기반 유사도 조인

유사한 시그니쳐 쌍을 찾아내기 위해 이전 단계에서 모인 시그니쳐들 사이의 편집 유사도를 계산하여 주어진 한계값 δ를 넘는 쌍을 찾는 문자 기반 유사도 조인을 수행한다. 이를 위한 관련 연구 중 가장 뛰어난 성능을 보인 맵리듀스 기반 문자열 유사도 조인[4]을 사용했다. 이 연구에서는 편집 거리를 유사도 기준으로 사용하는데 두 문자열 x와 y사이의 편집 유사도 한계값이 δ라면 다음과 같은 식을 통해 편집 거리 한계값 λ로 바꿀 수 있다.

$$ES(x,y) = 1 - \frac{ED(x,y)}{max(|x|,|y|)} \geq \delta \qquad \text{식 (3-9)}$$

$$1 - \frac{ED(x,y)}{|x|/\delta} \geq \delta \qquad \text{식 (3-10)}$$

$$ED(x,y) \leq \frac{|x| - \delta|x|}{\delta} = \frac{1-\delta}{\delta}|x| = \lambda \qquad \text{식 (3-11)}$$

이 때 비교할 문자열 x와 y사이의 편집거리가 λ이하라면 문자열 x를 $\lambda+1$개로 나누었을 때 그 중 반드시 한 조각은 나머지 문자열 y의 일부분과 일치한다. 그렇기에 모든 문자열을 여러 조각으로 나눈 후, 그 조각들을 각 문자열에 일치하는 부분이 있는지 확인해본다. 이 때 일치하는 부분이 있을 경우, 두 문자열 사이의 편집 유사도를 직접 계산하여 검증한다. 이 알고리즘은 모인 시그니쳐들 사이에서 유사한 쌍을 찾아내기 위함이기 때문에 자가 조인(self join)알고리즘이다. 이 알고리즘은 두 번의 맵리듀스 단계가 필요하고 의사코드는 그림10에 나타냈으며 자세한 방법은 다음과 같다.

Map단계에서는 주어진 시그니쳐를 $\lambda+1$개의 조각으로 나누어 각 조각을 키로 하고, 값에는 조각의 시작점과 전체 문자열의 길이 등 전지작업(pruning)에 필요한 정보들과 시그니쳐를 포함하고 있는 레코드의 정보를 함께 넣어 보내준다. 그리고 다시 한 번 주어진 시그니쳐에 대하여 키-값 쌍을 생성하는데 이 때 생성하는 키-값 쌍들은 앞에서 $\lambda+1$로 나눈 조각들과 일치할 가능성이 있는 문자열의 일부분을 후보(candidate)로 내보낸다. 그리고 마찬가지로 값에는 조각의 시작점과 전체 문자열의 길이 등 전지 작업에 필요한 정보들과 시그니쳐를 포함하고 있는 레코드의 정보를 함께 넣어 보내준다.

Reduce단계에서는 같은 키로 모인 키-값 쌍들을 모아 각 시그니쳐 쌍 사이의 편집 유사도가 각각 주어진 한계값 조건을 만족하는지 직접 계산해본다. 만약 시그니쳐 쌍이 유사하다면 시그니쳐 쌍을 키로 하고, 해당 시그니쳐를 포함하는 테이블R의 레코드 리스트와 테이블S의 레코드 리스트를 모아 값으로 하여 내보낸다. 이 때 중복이 생길 수가 있으므로 중복 제거를 위한 MapReduce작업이 한 번 더 필요하다. 중복 제거 작업에서의 키와 값은 문자 기반 유사도 조인 알고리즘의 결과로 나온 키와 값을 그대로 사용해서 모인 값들 중에 하나만 결과로 내보내는

방법으로 중복 결과를 제거한다.

문자 기반 유사도 조인 알고리즘을 실행하게 되면 그 결과는 그림9와 같다. 이때 (tv, tv)와 같이 일치하는 시그니쳐 쌍에 대한 레코드 리스트는 유사도 조인을 하지 않고도 얻을 수 있기에 구하지 않아도 된다.

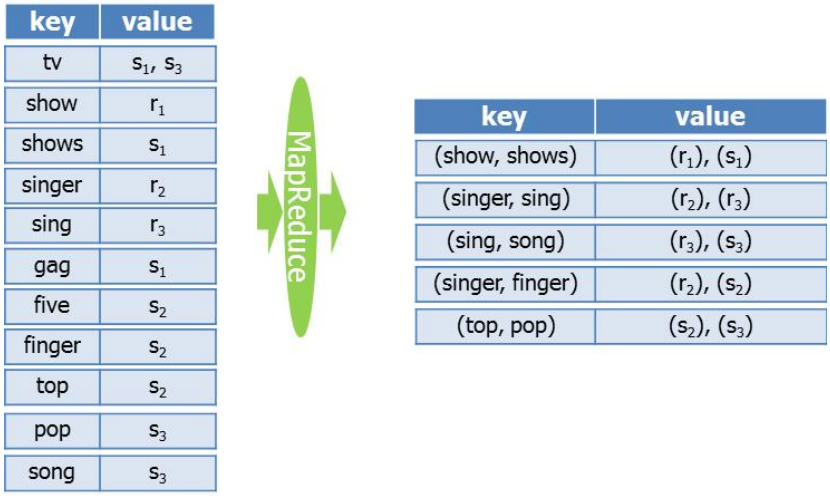

그림 9 문자 기반 유사도 조인 알고리즘의 예제

```
Function Ed-join.map (<t, P>)
begin
 1. T[λ+1]←partition t into (λ+1) segments
 2. for i = 1 : λ+1 do
 3.       emit(T[i],(i,|t|,"X",t,P))
 4. for L = lower_bound : upper_bound do
 5.       for j = 0 : |t| - L do
 6.            emit<(t[j:j+L-1]), (j,|t|,"Y",t,P)>
end
Function Ed-join.reduce(<signature, list(condition, token, list of
record)>)
begin
 1. Split list of value into list of "X" and list of "Y"
 2. for t1 ∈ list of "X" do
 3.       for t2 ∈ list of "Y" do
 4.            if(edit_similarity(t1,t2) ≥ δ)
 5.                 emit<(t1,t2), list of record>)
end

Function Rmd.map(<token pair(t1,t2), (list of r, list of s)>)
 1. emit<token pair(t1,t2), (list of r, list of s)>
Function Rmd.reduce(<token pair(t1,t2), list(list of r, list of s)>)
 1. emit<token pair(t1,t2), (list of r, list of s)>
```

그림 10 문자 기반 유사도 조인 알고리즘의 의사코드

제 4 절 작업 분배

유사한 시그니쳐 쌍을 찾은 후에는 이를 시그니쳐로 내보낸 레코드 쌍들 간의 유사도를 실제로 계산해보고 검증하여야 한다. 하지만 이 때 어떤 특정한 시그니쳐를 공통으로 갖고 있는 레코드들이 많다면 레코드 쌍의 유사도를 계산하여 검증하는 과정에서 연산량이 한대의 컴퓨터로 몰릴 수가 있다. 이를 막기 위해 하나의 시그니쳐 쌍에서 검증해야 하는 레코드 쌍의 수를 제한하여 max로 정해놓은 수를 초과한다면 레코드의 리스트를 분할하여야 한다. 그림11을 보면 첫 번째 시그니쳐 쌍과 두 번째 시그니쳐 쌍을 내보낸 레코드 쌍의 수는 각각 25개, 1개밖에 되지 않아 유사도 검증이 필요한 횟수가 매우 적지만 세 번째 시그니쳐 쌍을 보면 포함하고 있는 레코드 쌍의 수가 10000개가 되어 유사도 검증의 연산량이 많이 몰려있는 것을 알 수 있다. 이를 한 컴퓨터에서 계산하는 것을 막기 위해 그림11의 결과와 같이 나눠주고자 한다.

입력으로 받은 $list\ of\ r$ 과 $list\ of\ s$ 를 각각 LR 과 LS 라고 할 때, 레코드 리스트 LR 과 LS 를 조인한다면 만들어지는 레코드 쌍의 수는 $|LR| \cdot |LS|$개가 된다. 이를 컴퓨터의 숫자 M개로 분할하여 나누어 주고자 한다. LR을 x개로 분할하고 LS를 y개로 분할한다고 생각할 때 $x \cdot y = M$ 이 되어야 한다. 이 때 LS 를 y 개로 분할하였기에 조인 결과를 올바르게 계산하기 위해서는 같은 내용의 LR을 y번 복사해서 여러 대의 컴퓨터로 나눠주어야 하고 같은 이유로 LS 도 역시 x 번의 복사가 필요하다. 그렇다면 검증 단계에 입력으로 들어가는 레코드의 수는 원래 $|LR| + |LS|$ 개였지만 레코드 리스트를 분할하게 되면서 $y \cdot |LR| + x \cdot |LS|$ 개로 늘어나고 이를 최소화 하기 위한 x와 y를 구해보면 다음과 같다.

minimize $y \cdot |LR| + x \cdot |LS| \qquad s.t.\ x \cdot y = M$ 식 (3-12)

$$y \cdot |LR| + x \cdot |LS| = \frac{M \cdot |LR|}{x} + x \cdot |LS| \quad \text{식 (3-13)}$$

$y \cdot |LR| + x \cdot |LS|$ 은 $\frac{M \cdot |LR|}{x} = x \cdot |LS|$ 일 때 최소값을 가지게 되며 그때의 x와 y를 계산해보면 아래와 같다.

$$x = \sqrt{M \cdot |LR|/|LS|} \quad \text{식 (3-14)}$$

$$y = \sqrt{M \cdot |LS|/|LR||} \quad \text{식 (3-15)}$$

그러므로 LR 을 $\sqrt{M \cdot |LR|/|LS|}$ 개로 분할하고 LS 를 $\sqrt{M \cdot |LS|/|LR|}$ 개로 분할한다. 이를 통해 하나의 컴퓨터에 검증할 r과 s의 쌍이 몰리는 것을 방지한다.

Map단계에서는 하나의 시그니쳐 쌍에 몰린 레코드 쌍의 개수가 max 보다 큰지 확인해보고 크다면 r 과 s 의 레코드 리스트를 위에서 계산한 개수로 분할하여 내보내는데 이 때 레코드의 리스트를 보낼 컴퓨터를 선택하기 위해 랜덤하게 컴퓨터 번호를 선택해 키로 하고 분할된 레코드의 리스트를 값으로 하여 내보낸다. Reduce단계에서는 받은 레코드들의 리스트를 그대로 결과로 내보내 저장한다.

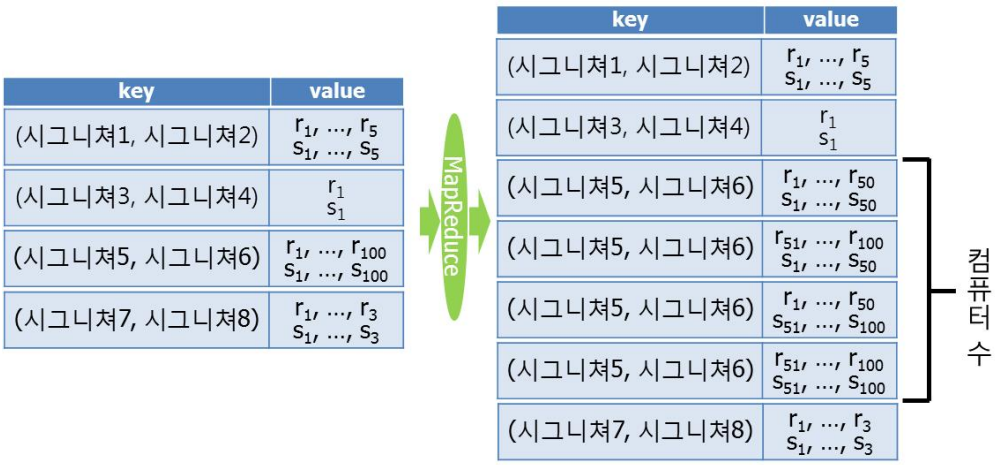

그림 11 작업 분배 알고리즘의 예제

작업 분배 알고리즘에 대한 의사코드는 그림12과 같다.

```
Function Load-bal.map(<token pair(t1,t2), (LR, LS)>)
begin
 1. If |LR| * |LS| > max then
 2.     Split each list of r and s into several segments
 3.     x=√(reducer · |LR|/|LS|),
 4.     y=√(reducer · |LS|/|LR|),
 5.     RL1…RLx ← LR
 6.     SL1…SLy ← LS
 7.     for 1≤i≤x do
 8.         for 1≤j≤y do
 9.             randomly select a computer ID
10.             emit<computer ID, (RLi,SLj)>
end

Function Load-bal.reduce(<reducer ID, list of (RL, SL)>)
begin
 1. for all (RL, SL) ∈ list(RL, SL) do
 2.     emit<null, (RL, SL)>
end
```

그림 12 작업 분배 알고리즘의 의사 코드

제 5 절 검증

모든 후보 레코드 쌍 사이의 유사도를 실제로 계산하고 걸러내는 단계이다. 이 때 $\theta \cdot |tr| \leq |ts| \leq |tr|/\theta$라는 범위에서만 유사할 가능성이 있다는 것을 식(3-4)와 식(3-7)을 통해 이미 알고 있으므로 불필요한 연산을 막기 위해 레코드 r과 레코드 s의 토큰 수를 비교하여 유사도 한계값 θ를 넘을 수 없는 쌍을 찾아 미리 걸러낼 수 있다. 이렇게 걸러낸 모든 쌍에 대하여 Fuzzy 토큰 자카드 유사도를 계산하고 유사도 한계값

```
Function Verification.map(<RL, SL>)
begin
 1. for r ∈ RL do
 2.     for s ∈ SL do
 3.         tr←tokenize r
 4.         ts←tokenize s
 5.         if θ|ts| ≤ |tr| ≤ |ts|/θ then
 6.             calculate exact similarity between r and s
 7.             if similarity≥θ then
 8.                 emit<(r,s), similarity>
end

Function Verification.reduce(<(r, s), list(similarity)>)
begin
 1. emit<(r,s), similarity>
end
```

그림 13 검증 알고리즘의 의사코드

θ를 넘는다면 결과로 내보낸다. 검증 알고리즘의 의사코드는 그림 13와 같다.

Map단계에서 위의 과정을 모두 실행하고 그대로 내보내며 Reduce 단계에서는 같은 키로 모인 값들이 모두 같은 값이기 때문에 그 중 하나만 내보내면 자연스럽게 중복 제거가 가능해진다. 그림14의 예제를 통해 확인해보면 총 세 개의 시그니쳐 쌍에 모인 모든 레코드 쌍에 대해 직접 유사도를 계산해보고 주어진 한계값 0.8을 넘는 (r_2, s_2)만을 유사도 조인의 결과로 내보낸다.

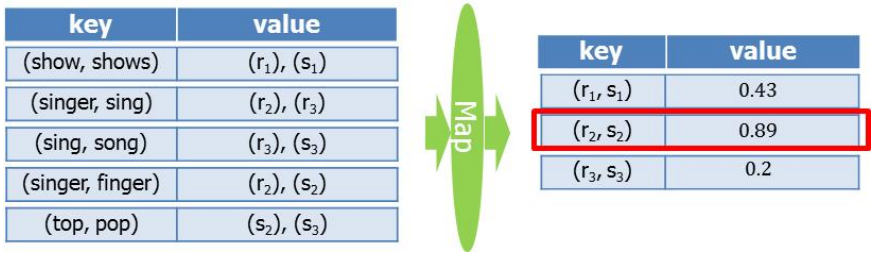

그림 14 검증 알고리즘의 예제

제 4 장 실험 및 결과

본 실험은 1대의 마스터 컴퓨터와 20대의 슬레이브 컴퓨터로 이루어진 클러스터를 이용해 수행했다. 마스터 컴퓨터는 Xeon 2.2GHz, 10MB cache, 8GB RAM, 1TB 7.2K RPM HDD의 성능을 가지며, 슬레이브 컴퓨터는 i3 3.3GHz, 3MB cache, 4GB RAM, 250GB 7.2K RPM HDD의 성능을 갖고 있다. 본 논문에서는 두 가지 종류의 실험을 수행하였다. 첫 번째 실험으로는 단일 머신 알고리즘[1]과 본 논문의 알고리즘이 같은 유사도 조인을 했을 때 수행시간이 어느 정도 차이가 나는지에 대한 실험을 진행하였다. 그리고 두 번째로는 분산처리 유사도 조인 알고리즘의 효율성을 알아보기 위하여 큰 데이터에 대해 사용하는 컴퓨터의 수를 바꿔가면서 수행 시간을 측정하는 실험을 진행하였다. 본 논문의 알고리즘은 Java로 구현되었고 Hadoop-1.2.1버전을 사용해 실험을 수행하였다. 그리고 실험데이터는 AOL 검색 쿼리 데이터[6]를 사용하였다.

제 1 절 단일 머신 알고리즘과의 비교

본 논문의 알고리즘이 단일 머신 알고리즘[1]과 비교했을 때 어느 정도의 성능을 보이는지 알기 위한 실험을 수행하였다. 이 실험에서는 같은 유사도를 사용한 단일 머신 알고리즘[1]과 본 논문의 분산 처리 유사도 조인 알고리즘의 수행시간을 비교하는 방식으로 진행하였으며 단일 머신의 알고리즘은 C++으로 구현되어 gcc 4.2.3으로 컴파일 된 실행파일을 다운 받아 사용하였다[5]. 그리고 사용한 데이터의 레코드 개수는 2,417,288개이며 용량은 72MB이다. 실험에 필요한 편집 유사도의 한계값 δ은 0.8을 사용하였고 유사도의 한계값 θ는 0.7부터 0.9까지 바꿔가면서 실험을 수행하였다. 그 결과는 다음의 그래프와

같다.

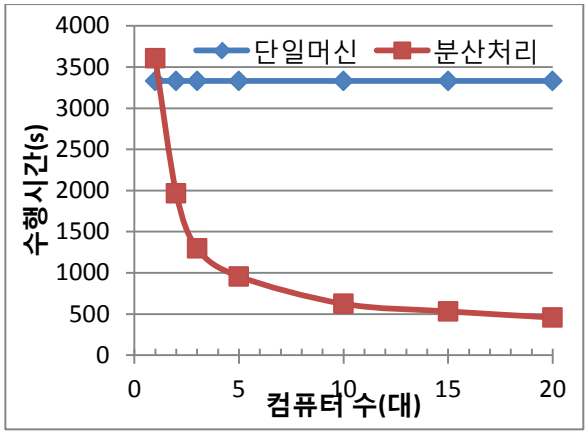

그림 15 수행시간 비교(유사도 한계값 0.7)

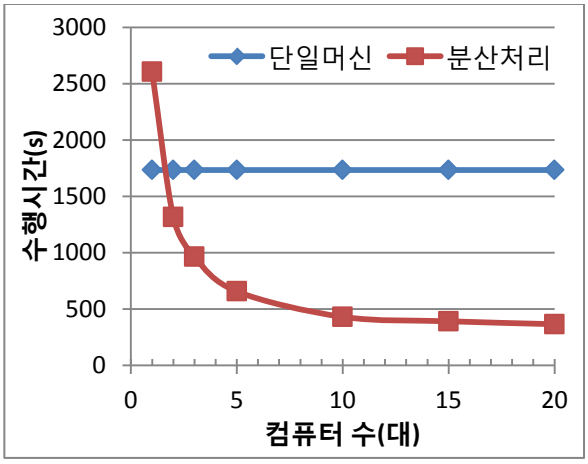

그림 16 수행시간 비교(유사도 한계값 0.8)

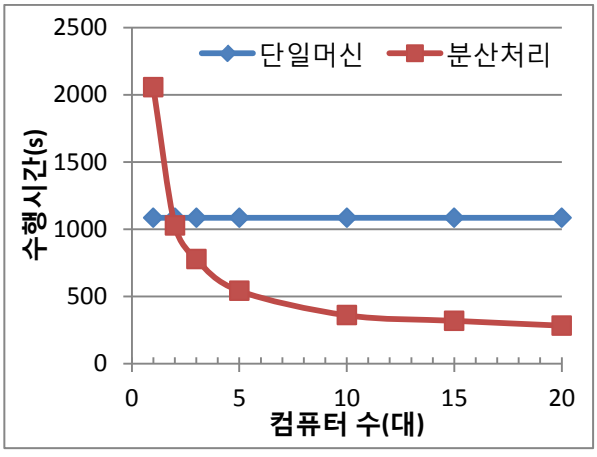

그림 17 수행시간 비교(유사도 한계값 0.9)

분산 처리 유사도 조인 알고리즘에서 사용하는 슬레이브 컴퓨터의 수를 1대부터 20대까지 바꿔가면서 실험한 결과, 한대의 컴퓨터를 이용했을 때는 모두 단일 머신 알고리즘이 더 수행시간이 적게 나타났다. 이는 맵리듀스 프레임워크를 사용함으로써 발생하는 추가 비용 때문으로 해석할 수 있다. 하지만 두 대의 컴퓨터를 이용했을 때부터 분산처리 알고리즘이 수행시간이 더 적게 걸리기 시작했으며 컴퓨터를 늘려갈수록 분산 처리 유사도 조인 알고리즘이 수행 속도가 향상되는 것을 그림15~17에서 확인할 수 있다. 20대를 모두 사용했을 때는 분산 처리 유사도 조인 알고리즘이 단일 머신 알고리즘에 비해 적게는 3.8배부터 최대 7.3배까지 수행 속도가 빨라지는 것을 확인할 수 있었다.

또한 이 논문에서는 새로운 시그니쳐를 제안하였기에 단일 머신 알고리즘에 제안하는 시그니쳐보다 얼마나 효율적으로 레코드 쌍을 걸러내는지에 대한 실험을 수행하였다. 데이터는 수행 시간 비교에서와 같은 데이터를 사용하였고 편집 유사도 한계값 δ은 0.8을 사용하였고 유사도의 한계값 θ도 0.8로 하였을 때 시그니쳐를 이용해 걸러낸 후 직접 검증하는 후보 레코드 쌍의 수를 세어보았다. 그 결과 그림 18에서와 같이 단일 머신 알고리즘은 약 30억개 레코드 쌍에 대해서 실제로 유사도를 계산했지만, 제안한 분산 처리 유사도 조인 알고리즘은 그의 5% 정도인 1억 6천만 개만의 레코드 쌍에 대해서만 직접 유사도를 계산하였다. 따라서 제안한 분산 처리 유사도 조인 알고리즘의 효율성은 분산 병렬 처리를 했기 때문이기도 하지만, 실질적인 유사도 계산을 더 적게 하기 때문임을 알 수 있다.

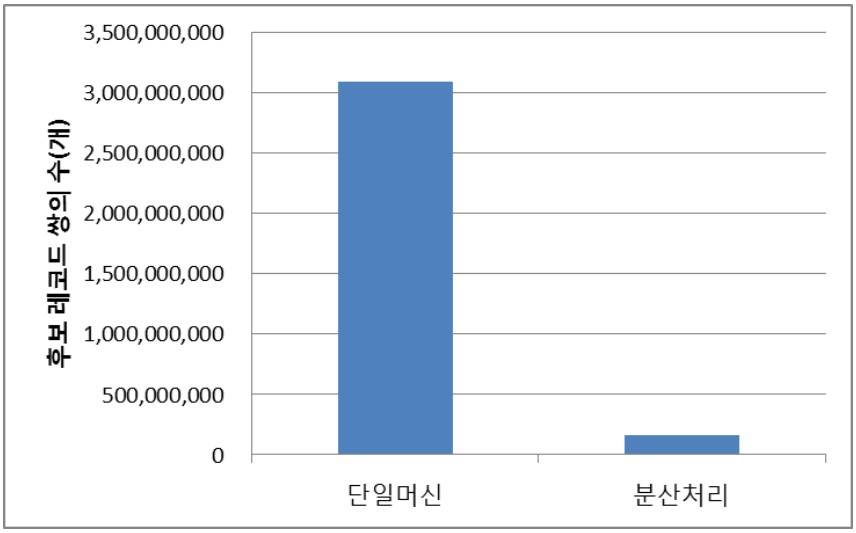

그림 18 시그니쳐 성능 비교

제 2 절 컴퓨터 수에 따른 수행시간 및 효율

본 논문의 알고리즘은 여러 대의 컴퓨터를 사용해 수행 시간을 향상시키기 위한 분산 처리 알고리즘이다. 따라서 컴퓨터의 수를 늘려감에 따라 비례하여 성능이 향상되는지를 확인해보아야 한다. 이 실험에서는 32,000,000개의 레코드를 포함하고 있는 1GB의 데이터를 사용했으며 컴퓨터의 개수를 5대에서 20대까지 바꿔가며 실험을 수행하였다. 실험에 필요한 편집 유사도의 한계값 δ은 0.8부터 0.9까지 변화시켜 주었으며 유사도의 한계값 θ은 0.7부터 0.9까지 변화시키며 실험을 수행하였다. 결과는 유사도 한계값에 따른 수행시간에 대한 그래프와 수행 시간 결과를 5대의 컴퓨터를 이용해 수행한 실험의 시간으로 나눈 값을 나타낸 속도비율 그래프가 있다. 단일 머신 알고리즘은 생성한 시그니쳐를 모두 메모리에 올려야 하는 알고리즘이기에 큰 데이터에 대해서는 실행이 불가능하기에 같이 비교하지 못하였다.

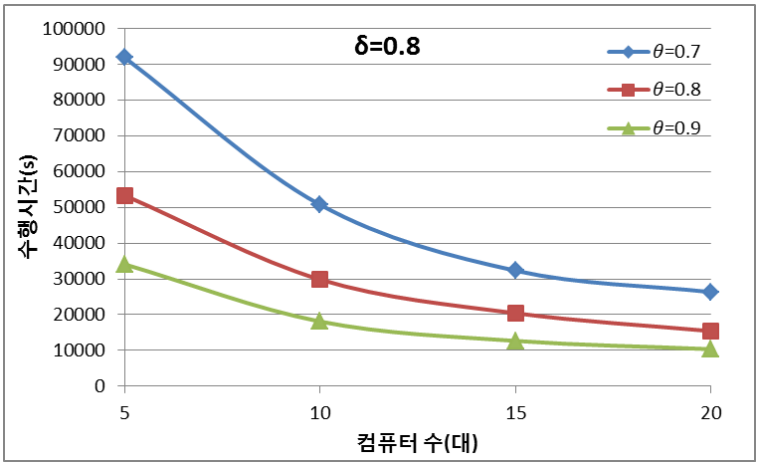

그림 19 분산 처리 알고리즘의 수행시간(편집유사도 한계값 0.8)

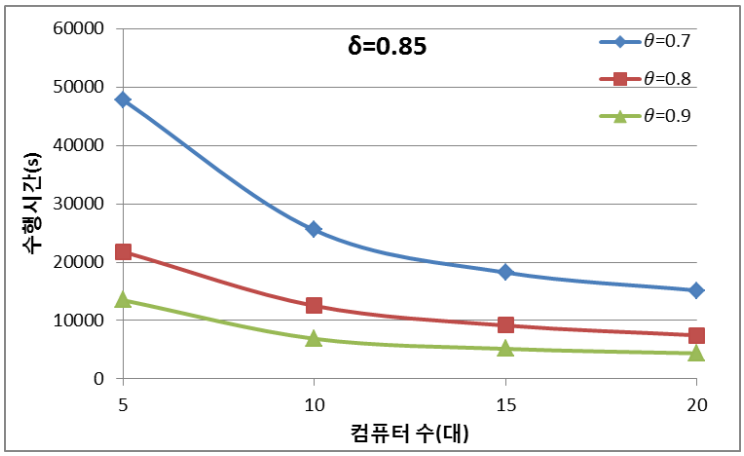

그림 20 분산 처리 알고리즘의 수행시간(편집유사도 한계값 0.85)

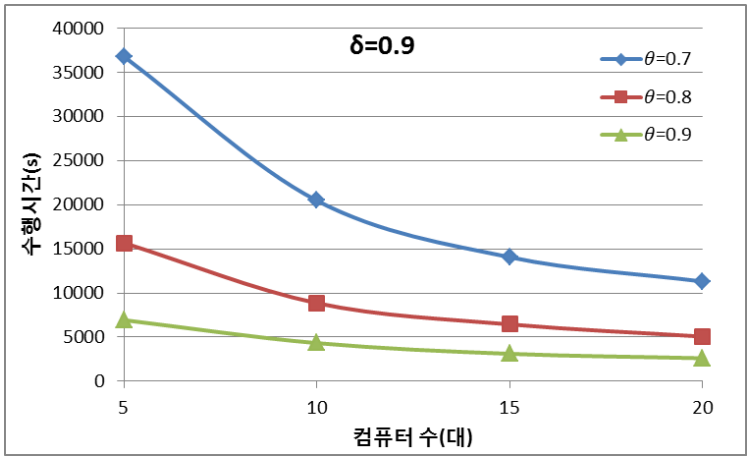

그림 21 분산 처리 알고리즘의 수행시간(편집유사도 한계값 0.9)

그림 19~21는 편집 유사도의 한계값 δ과 유사도의 한계값 θ에 따른 유사도 조인의 수행시간을 보여준다. 편집 유사도의 한계값이 높아질수록 비슷한 시그니쳐의 수가 줄기 때문에 수행 시간 역시 줄어드는 것을 확인할 수 있었으며 유사도의 한계값은 생성되는 시그니쳐의 수에 직접적인 영향을 끼치기 때문에 역시 수행 시간이 줄어드는 것을 확인할 수 있다. 또한 많은 수의 컴퓨터를 사용할수록 수행시간이 급격하게 줄어드는 것을 확인할 수 있는데 얼마나 수행 속도를 향상시키는지를 그림 22~24에 나타내었다.

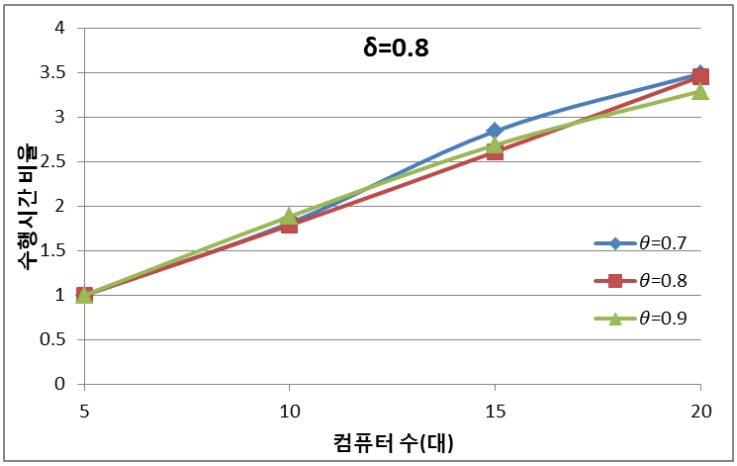

그림 22 컴퓨터 수에 따른 수행 시간 비율(편집유사도 한계값 0.8)

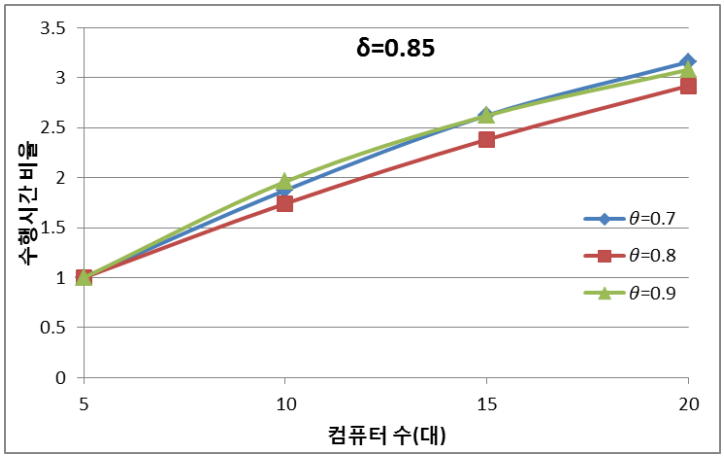

그림 23 컴퓨터 수에 따른 수행 시간 비율(편집유사도 한계값 0.85)

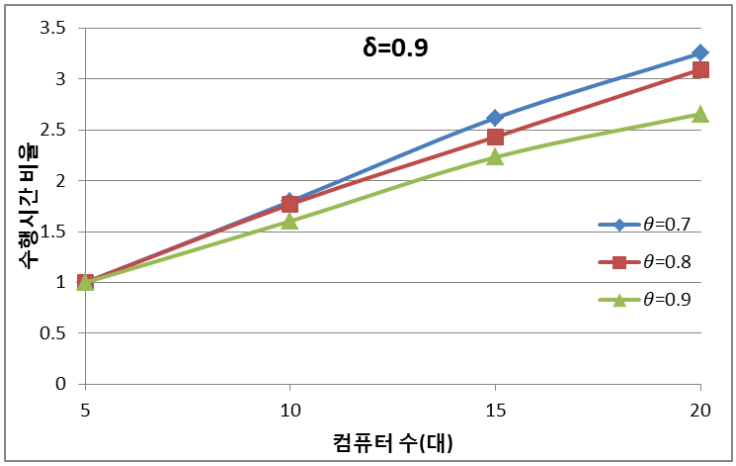

그림 24 컴퓨터 수에 따른 수행 시간 비율(편집유사도 한계값 0.9)

그림 22~24에서는 컴퓨터의 수를 늘렸을 때 몇 배나 수행 속도가 증가하는지를 나타냈다. 가로축은 사용한 컴퓨터의 개수, 세로축은 각 수행시간을 컴퓨터 5대를 사용했을 때의 수행시간으로 나눈 값이다. 이렇게 그래프를 그릴 때, 컴퓨터 수를 늘린다면 10대를 사용했을 때는 2배로 빨라져야 하며 15대일때는 3배, 20대일때는 4배가 나오는 것이 가장 이상적인 경우이다. 그래프를 확인해보았을 때, 이상적인 경우에 가깝게 수행시간 비율이 나타났으며 컴퓨터 수를 늘릴수록 거의 사용한 컴퓨터의 수에 비례하게 선형적으로 성능이 향상되는 것을 확인할 수 있다. 따라서 이 알고리즘은 분산 처리 효율이 매우 뛰어난 알고리즘이라는 것을 알 수 있다.

제 5 장 결론

본 논문에서는 Fuzzy 토큰 자카드 유사도를 이용한 효율적인 문자열 유사도 조인을 위한 분산 처리 알고리즘을 제안하였다. 문자열의 일부분만을 내보내는 새로운 시그니쳐 방법을 제안하여 후보 레코드 쌍을 줄임과 동시에 유사도 연산량을 줄여 속도를 향상시켰으며 큰 용량의 데이터에서 한대의 컴퓨터에 작업량이 몰리는 것을 막기 위한 데이터를 분배하는 방법을 포함하는 총 5단계로 이루어진 분산처리 알고리즘을 제안하였다. 그리고 실험을 통해 단일 머신 알고리즘에 비해 그 성능이 뛰어남을 확인할 수 있었다. 또한 여러 대의 컴퓨터를 사용할수록 성능이 선형에 가깝게 증가하는 효과적인 분산 처리 알고리즘임을 확인할 수 있었다.

참고 문헌

[1] J. Wang, G. Li and J. Fe, “Fast-Join: An Efficient Method for Fuzzy Token Matching based String Similarity Join”, In ICDE, 2011.

[2] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters”, In OSDI, 2004.

[3] G. Li, D. Deng, J. Wang and J. Feng, “Pass-Join: A Partition-based Method for Similarity Joins”, In VLDB, 2011

[4] D. Deng, G. Li, S. Hao, J. Wang and J. Feng, “MassJoin: A MapReduce-based Method for Scalable String Similarity Joins”, In ICDE, 2014

[5]Fast join algorithm, http://www.cs.berkeley.edu/~jnwang/codes/fastjoin.tar.gz

[6] aol search query data, http://www.cim.mcgill.ca/~dudek /206/Logs/AOL-user-ct-collection/

[7] Apache Hadoop, https://hadoop.apache.org/

[8] Harold W. Kuhn, "The Hungarian Method for the assignment problem", *Naval Research Logistics Quarterly*, 2: 83–97, 1955

[9] Okcan, Alper, and Mirek Riedewald. "Processing theta-joins using MapReduce." *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011.

[10] Zhang, Xiaofei, Lei Chen, and Min Wang. "Efficient multi-way theta-join processing using mapreduce." *Proceedings of the VLDB Endowment* 5.11 (2012): 1184-1195.

[11] Blanas, Spyros, et al. "A comparison of join algorithms for log processing in mapreduce." *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010.

[12] Vernica, Rares, Michael J. Carey, and Chen Li. "Efficient parallel set-similarity joins using MapReduce." *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010.

[13] Kim, Younghoon, and Kyuseok Shim. "Parallel top-k similarity join algorithms using MapReduce." *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012.

[14] Metwally, Ahmed, and Christos Faloutsos. "V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors."*Proceedings of the VLDB Endowment* 5.8 (2012): 704-715.

[15] Baraglia, Ranieri, Gianmarco De Francisci Morales, and Claudio Lucchese. "Document similarity self-join with mapreduce." *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010.

[16] Elsayed, Tamer, Jimmy Lin, and Douglas W. Oard. "Pairwise document similarity in large collections with MapReduce." *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*. Association for Computational Linguistics, 2008.

[17] Park, Yoonjae, Jun-Ki Min, and Kyuseok Shim. "Parallel computation of skyline and reverse skyline queries using mapreduce.", VLDB, 2013.

[18] Kim, Younghoon, et al. "DBCURE-MR: an efficient density-based clustering algorithm for large data using MapReduce." *Information Systems* 42 (2014): 15-35.

[19] Kim, Younghoon, and Kyuseok Shim. "TWITOBI: A recommendation system for twitter using probabilistic modeling." Data Mining (ICDM), 2011 IEEE 11th International Conference on. IEEE, 2011.

[20] Kyuseok Shim: MapReduce Algorithms for Big Data Analysis. PVLDB 5(12): 2016-2017 (2012)

Abstract

Efficient String Similarity Joins using MapReduce

Changhyung Lee
Electrical and Computer Engineering
The Graduate School
Seoul National University

String similarity joins are very important and used frequently in the database area. Recently, the similarity measure based on fuzzy-token matching was proposed to take the advantages of token-based as well as character-based similarities. However, it is very expensive to perform similarity joins for big data. To improve the performance of similarity joins, we first propose a parallel algorithm using MapReduce framework which utilizes our new signature. We next conduct the performance study with the state-of-the-art serial algorithm on a single machine by experiments. Our MapReduce algorithm with 20 machines is 7 times faster than the serial algorithm. We show that the more machines we use, the shorter execution time our algorithm takes.

Keywords : String, Similarity join, MapReduce, Algorithm, Hadoop
Student Number : 2013-20864